UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

Optimal Control of a bipedal robot

Professor: **Giuseppe Notarstefano**

Students:Ange derick Nzangue 0001044659
Mahdi Hamadi 0001030867
Foasse Nyie Richard Duval 0001059154

Academic year 2022/2023

Abstract

In the following project we will explore how a simplified dynamical model of a biped robot can be optimally controlled using ***Newton's algorithm*** for optimal control in order to move from one equilibrium point to another and to perform a certain desired trajectory in particular a smooth reference trajectory to perform a step ahead.

One goal that emerges from the studies of bipedal robot is try to reproduce as much as possible the human walking.Construct a biped robot capable of walking on different types of ground is a major challenge. In the literature, there are several studies and methods for the design of effective control laws to stabilize the walking of a biped robot and a common point between these studies is the tracking of some desired reference trajectory, which should be designed. A design of reference trajectories for the walking cycles of the robot is important and not trivial [Bertsekas(1997)]. In rest part of this project, we explore, design and tracking some trajectory for our robot and simulate its behavior.

Contents

Introduction

Motivations

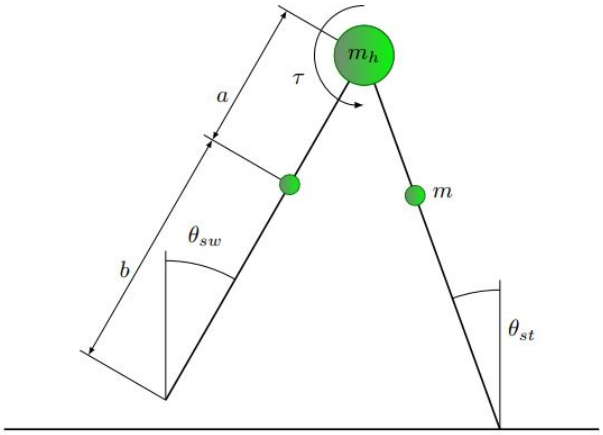
Optimal Control M, taught by Professor *Notarstefano*, has examined a variety of optimal control methods and tactics to regulate the dynamical system behavior. Therefore, to reinforce the general learning process, we are required to create a project at the conclusion of the classes that involves analyzing and regulating a dynamical system using the control techniques we have learned during the course.

Contributions

Our project was developed under the direction and supervision of Simone Baroncini, our instructor, to whom we are especially grateful.

Chapter 1

Definition of the dynamic model - Task 0



1.1 Walking robot dynamic

For a simplified walking robot, often known as a compass gait model, we have created an optimal control approach in this study. In this model, the dynamics of both the stance leg and the swing leg are taken into account. With a pin joint at the hip, it has two spokes. In this part we have to rewrite our model equation in state-space form and discretize it.
So we describe the state of the robot with four state variables: $\theta_{sw}, \theta_{st}, \dot{\theta}_{sw}, \dot{\theta}_{st}$, where the abbreviation st is shorthand for the stance leg and sw for the swing leg. Denote $q = [\theta_{sw}, \theta_{st}]^\top$ and $u = \tau$. The dynamics of the compass leg is given by:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu,$$

with

$$M(q) = \begin{bmatrix} mb^2 & -mlb\cos(\theta_{st}-\theta_{sw}) \\ -mlb\cos(\theta_{st}-\theta_{sw}) & (m_h+m)\,l^2 + ma^2 \end{bmatrix} \quad G(q) = \begin{bmatrix} mbg\sin\theta_{sw} \\ -(m_h l + ma + ml)\,g\sin\theta_{st} \end{bmatrix}$$

$$C(q,\dot{q}) = \begin{bmatrix} 0 & mlb\sin(\theta_{st}-\theta_{sw})\,\dot{\theta}_{st} \\ mlb\sin(\theta_{st}-\theta_{sw})\,\dot{\theta}_{sw} & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

.

By defining the state vector and the input torque as: [Ames and Poulakakis(2018)]

$$q = \left[\theta_{sw}, \theta_{st}, \dot{\theta}_{sw}, \dot{\theta}_{st}\right]^\top = [x_1, x_2, x_3, x_4]^\top$$
$$u = \tau$$

Since the matrix M(q) is invertable we can rewrite first the system form as follows :

$$\ddot{q} = -M(q)^{-1}C(q,\dot{q})\dot{q} + M(q)^{-1}G(q) + M(q)^{-1}Bu,$$

where

$$M(q)^{-1} = \frac{1}{z(\theta_{st},\theta_{sw})}\begin{bmatrix} (m_h+m)\,l^2 + ma^2 & mlb\cos(\theta_{st}-\theta_{sw}) \\ mlb\cos(\theta_{st}-\theta_{sw}) & mb^2 \end{bmatrix}$$

with

$$z(\theta_{st},\theta_{sw}) = mb^2((m_h+m)l^2 + ma^2) - (mlb\cos(\theta_{st}-\theta_{sw}))^2$$

$$l = a + b,\ m = 0.2 \text{ kg},\ m_h = 2 \text{ kg},\ a = 0.5 \text{ m},\ b = 0.7 \text{ m} \tag{1.1}$$

1.2 State space model of the walking robot dynamic

State-space model

The state variables were placed in a 4-dimensional vector **x** and the control inputs in a 1-dimensional vector **u**, both of which were described as follows, to cast our model in a generic state-space framework.
where:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \theta_{sw} \\ \theta_{st} \\ \dot{\theta}_{sw} \\ \dot{\theta}_{st} \end{bmatrix} \tag{1.2}$$

$$\begin{cases} \dot{x}_1 = x_3 \\ \dot{x}_2 = x_4 \\ \dot{x}_3 = (-\left(s_1^2\right)\cos(x_2-x_1)\sin(x_2-x_1)\,x_3^2 + s_2 s_1 \sin(x_2-x_1)\,x_4^2 \\ \qquad\qquad +(-s_2 s_3 \sin(x_1) - s_1 s_4 \cos(x_2-x_1)\sin(x_2)) + u(s_2 - s_1\cos(x_2-x_1)))/\text{den} \\ \dot{x}_4 = (-\left(s_1 s_2 \sin(x_2-x_1)x_3^2 + (s_1^2)\cos(x_2-x_1)\sin(x_2-x_1)x_4^2\right) \\ \qquad\qquad + (-(s_1 s_3 \cos(x_2-x_1)\sin(x_1) - s_2 s_4 \sin(x_2))) + u(-s + s_1\cos(x_2-x_1)))/\text{den} \end{cases} \tag{1.3}$$

with:

$$den = (s * s_2 - s_1^2 * \cos(x_{2,t} - x_{1,t})^2)$$

$s = mb^2,\ s_1 = mlb,\ s_2 = (m_h+m)\,l^2 + ma^2,\ s_3 = mgb,\ s_4 = (ma + ml + m_h)\,g.$

1.3 Discretization

We use Euler method of discretization (also called the forward Euler method), that is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. we use it to transform our system into a form that can be used to calculate numerical solutions and take into account a limited time step dt that will be the inverse of the discrete time model's operating frequency.

$$x_{t+1} = x_t + \delta f(x_t, u_t) = x_t + dt * f(x_t, u_t) \tag{1.4}$$

$$\begin{cases} \mathbf{x}_{1,t+1} = x_{1,t} + \mathrm{dt} * x_{3,t} \\ \\ \mathbf{x}_{2,t+1} = x_{2,t} + \mathrm{dt} * x_{4,t} \\ \\ \mathbf{x}_{3,t+1} = x_{3,t} + \mathrm{dt} * \left(-\left(\left(s_1^2\right)\cos\left(x_{2,t} - x_{1,t}\right)\sin\left(x_{2,t} - x_{1,t}\right)x_{3,t}^2 + s_2 s_1 \sin\left(x_{2,t} - x_{1,t}\right)x_{4,t}^2\right) + \right. \\ \qquad \left. (-(s_2 s_3 \sin(x_{1,t}) - s_1 s_4 \cos(x_{2,t} - x_{1,t})\sin(x_{2,t})) + u(s_2 - s_1\cos(x_{2,t} - x_{1,t}))\right)/\mathrm{den} \\ \\ \mathbf{x}_{4,t+1} = x_{4,t} + \mathrm{dt} * \left(-\left(s_1 s_2 \sin(x_{2,t} - x_{1,t})x_{3,t}^2 + (s_1^2)\cos(x_{2,t} - x_{1,t})\sin(x_{2,t} - x_{1,t})x_{4,t}^2\right) + \right. \\ \qquad \left. (-(s_1 s_3 \cos(x_{2,t} - x_{1,t})\sin(x_{1,t}) - s_2 s_4 \sin(x_{2,t})) + u(-s + s_1\cos(x_{2,t} - x_{1,t}))\right)/\mathrm{den} \end{cases} \tag{1.5}$$

1.4 Computation of the first derivatives

In this part we must compute the gradient of our model to find the matrices of the dynamic. Where we construct the vectorial function of the derivatives with respect to the state x and input u.

$$\nabla_x f_i = \begin{bmatrix} \frac{\partial f_1}{\partial x_i} \\ \frac{\partial f_2}{\partial x_i} \\ \frac{\partial f_3}{\partial x_i} \\ \frac{\partial f_4}{\partial x_i} \end{bmatrix}^T = \begin{bmatrix} \nabla_x f_1 & \nabla_x f_2 & \nabla_x f_3 & \nabla_x f_4 \end{bmatrix} \tag{1.6}$$

$$\nabla_x f_1 = \begin{bmatrix} \nabla_{x1} f_1 \\ \nabla_{x2} f_1 \\ \nabla_{x3} f_1 \\ \nabla_{x4} f_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \delta \\ 0 \end{bmatrix} \qquad \nabla_x f_2 = \begin{bmatrix} \nabla_{x1} f_2 \\ \nabla_{x2} f_2 \\ \nabla_{x3} f_2 \\ \nabla_{x4} f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \delta \end{bmatrix} \tag{1.7}$$

$$\nabla_x f_3 = \begin{bmatrix} \nabla_{x1} f_3 \\ \nabla_{x2} f_3 \\ \nabla_{x3} f_3 \\ \nabla_{x4} f_3 \end{bmatrix} \qquad \nabla_x f_4 = \begin{bmatrix} \nabla_{x1} f_4 \\ \nabla_{x2} f_4 \\ \nabla_{x3} f_4 \\ \nabla_{x4} f_4 \end{bmatrix} \tag{1.8}$$

The equation of $\nabla_{x_i} f_3$ and $\nabla_{x_i} f_4$ are too long that cannot be shown here.

$$\nabla_u f_i = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial u} \\ \frac{\partial f_4}{\partial u} \end{bmatrix}^T = \begin{bmatrix} \nabla_u f_1 & \nabla_u f_2 & \nabla_u f_3 & \nabla_u f_4 \end{bmatrix} \tag{1.9}$$

$$\nabla_u f = \begin{bmatrix} 0 \\ 0 \\ \frac{\delta\big((a^2 \cdot m + l^2(m+m_h)) - blm\cos(x_2 - x_1)\big)}{(-b^2 l^2 m^2 \cos^2(x_2 - x_1) + b^2 m(a^2 m + l^2(m+m_h)))} \\ \frac{\delta\big(-b^2 \cdot m + blm\cos(x_2 - x_1)\big)}{(-b^2 l^2 m^2 \cos^2(x_2 - x_1) + b^2 m(a^2 m + l^2(m+m_b)} \end{bmatrix}^T \tag{1.10}$$

Chapter 2

Trajectory exploration:step ahead - Task1

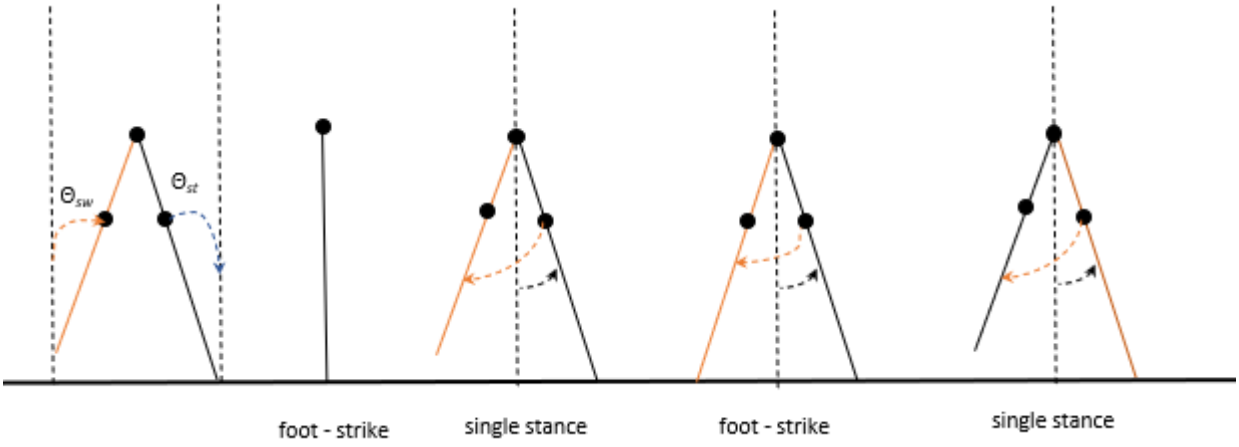


Figure 2.1: Diffrent configuration of robot leg.

Our goal in this part, is to understand the behavior of our dynamical system, in order to find the equilibrium positions or fixed points and after bring the robot from one position to another [Znegui et al.(2020a)Znegui, Gritli, and Belghith]. A fixed point represent the static postures for our robot in which it can stand safely. look for picture illustration 2.1.

A state $\bar{x}$ is saw to be a fixed point for our system if it satisfy our dynamics equations(2.5). We can see from picture illustration 2.2 that the robot stay in equilibrium when center of mass of the two legs are in the same position in the vertical axis.This frase is traducted by the follow equation:

$$\begin{cases} l * \cos\theta_{st} - \mathrm{b} * \cos\theta_{sw} = \mathrm{a} * \cos\theta_{st} \\ l * \cos\theta_{st} + \mathrm{a} * \cos\theta_{st} = \mathrm{b} * \cos\theta_{sw} \\ \cos\theta_{st} * (l - \mathrm{a}) = \mathrm{b} * \cos\theta_{sw} \\ \cos\theta_{st} = \cos\theta_{sw} \end{cases} \tag{2.1}$$

After seeing one of relation that lead the robot configuration we can start now to study the set of points that will define our equilibrium trajectories.

2.1 Equilibrium trajectories

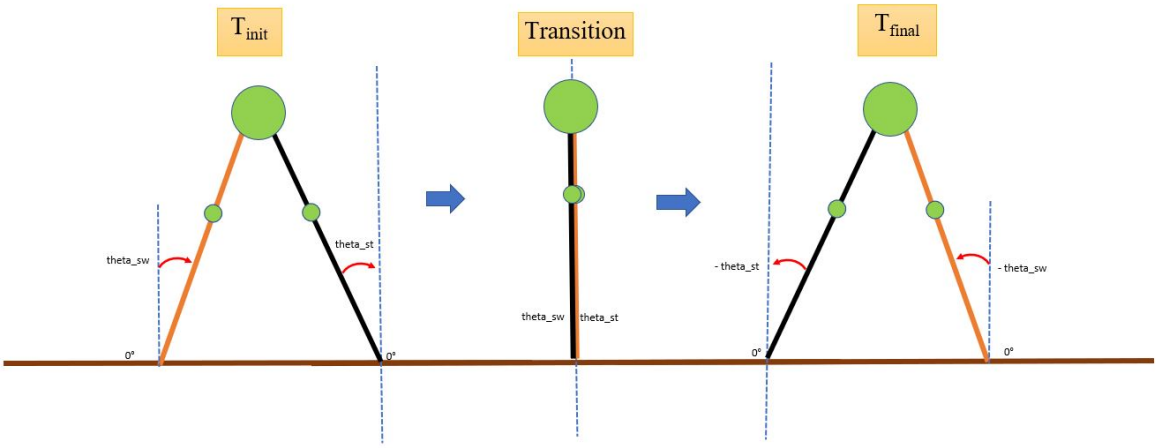


Figure 2.2: Compass gait transition between two equilibria

Starting with the discrete-time state-space model, we want to construct a desired trajectory, in order to compute its equilibrium trajectories, in which we expect to be fixed for all time. To make this, we substitute the coordinates of our suppose fixed point ($\bar{x}_t$) in the discret time system and obtain the following system of equations:

$$
\begin{cases}
\mathbf{x}_{1,t} = x_{1,t} + \mathrm{dt} * x_{3,t} \\
\\
\mathbf{x}_{2,t} = x_{2,t} + \mathrm{dt} * x_{4,t} \\
\\
\mathbf{x}_{3,t} = x_{3,t} + \mathrm{dt} * \left(- \left(\left(s_1^2 \right) \cos\left(x_{2,t} - x_{1,t} \right) \sin\left(x_{2,t} - x_{1,t} \right) x_{3,t}^2 + s_2 s_1 \sin\left(x_{2,t} - x_{1,t} \right) x_{4,t}^2 \right) + \right. \\
\qquad \left. \left(-(s_2 s_3 \sin(x_{1,t}) - s_1 s_4 \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t})) + u(s_2 - s_1 \cos(x_{2,t} - x_{1,t})) \right) / \mathrm{den} \right. \\
\\
\mathbf{x}_{4,t} = x_{4,t} + \mathrm{dt} * \left(- \left(s_1 s_2 \sin(x_{2,t} - x_{1,t}) x_{3,t}^2 + (s_1^2) \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t} - x_{1,t}) x_{4,t}^2 \right) + \right. \\
\qquad \left. \left(-(s_1 s_3 \cos(x_{2,t} - x_{1,t}) \sin(x_{1,t}) - s_2 s_4 \sin(x_{2,t})) + u(-s + s_1 \cos(x_{2,t} - x_{1,t})) \right) / \mathrm{den} \right.
\end{cases}
\tag{2.2}
$$

$$\begin{cases} 0 = \mathrm{dt} * x_{3,t} \\ \\ 0 = \mathrm{dt} * x_{4,t} \\ \\ 0 = \mathrm{dt} * \left(- \left(\left(s_1^2 \right) \cos\left(x_{2,t} - x_{1,t} \right) \sin\left(x_{2,t} - x_{1,t} \right) x_{3,t}^2 + s_2 s_1 \sin\left(x_{2,t} - x_{1,t} \right) x_{4,t}^2 \right) + \right. \\ \qquad \left. \left(-(s_2 s_3 \sin(x_{1,t}) - s_1 s_4 \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t})) + u(s_2 - s_1 \cos(x_{2,t} - x_{1,t})) \right) / \mathrm{den} \\ \\ 0 = \mathrm{dt} * \left(- \left(s_1 s_2 \sin(x_{2,t} - x_{1,t}) x_{3,t}^2 + (s_1^2) \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t} - x_{1,t}) x_{4,t}^2 \right) + \right. \\ \qquad \left. \left(-(s_1 s_3 \cos(x_{2,t} - x_{1,t}) \sin(x_{1,t}) - s_2 s_4 \sin(x_{2,t})) + u(-s + s_1 \cos(x_{2,t} - x_{1,t})) \right) / \mathrm{den} \end{cases} \tag{2.3}$$

$$\begin{cases} x_{3,t} = 0 \\ \\ x_{4,t} = 0 \\ \\ 0 = (-(s_2 s_3 \sin(x_{1,t}) - s_1 s_4 \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t})) + u(s_2 - s_1 \cos(x_{2,t} - x_{1,t}))) \\ 0 = (-(s_1 s_3 \cos(x_{2,t} - x_{1,t}) \sin(x_{1,t}) - s_2 s_4 \sin(x_{2,t})) + u(-s + s_1 \cos(x_{2,t} - x_{1,t}))) \end{cases} \tag{2.4}$$

Since we have just one input to control both leg, we need to choose the two angles to make this possible by solving two equation with three unknwon. but according to the first relation position saw before (2.1), we can fix one of the two angles and find the other two solving numerically the system since becomes two equation with two unknwon that allow us to find the second angle and the corresponding torque τ

$$\begin{cases} u = ((s_2 s_3 \sin(x_{1,t}) - s_1 s_4 \cos(x_{2,t} - x_{1,t}) \sin(x_{2,t})) / (s_2 - s_1 \cos(x_{2,t} - x_{1,t}))) \\ u = ((s_1 s_3 \cos(x_{2,t} - x_{1,t}) \sin(x_{1,t}) - s_2 s_4 \sin(x_{2,t})) / (-s + s_1 \cos(x_{2,t} - x_{1,t}))) \end{cases} \tag{2.5}$$

In order to automate this process, we write a code that solver this and restitute the two variables needded (Chosing θ_{sw}we find u and θ_{st}).

2.2 Type of trajectory

2.2.1 Transition as Step from one equilibrium to another

In order to create a trajectory between two equilibrium point, we used the previous condition to fix the first angle and generate the starting point of our desired state and input. After that we consider a change state that occur in the fixed time to generate the new state and input considering a step model as :

$$2 * l * \sin\left(0.5 * (\theta_{st} + \theta_{sw})\right) \tag{2.6}$$

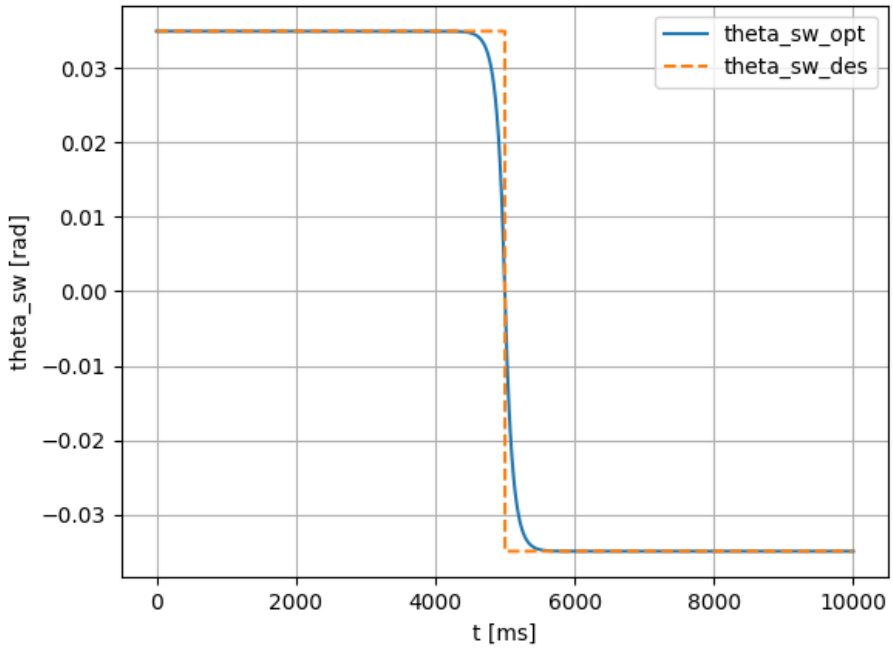


Figure 2.3: Step transition between θ_{sw_1} and θ_{sw_2}

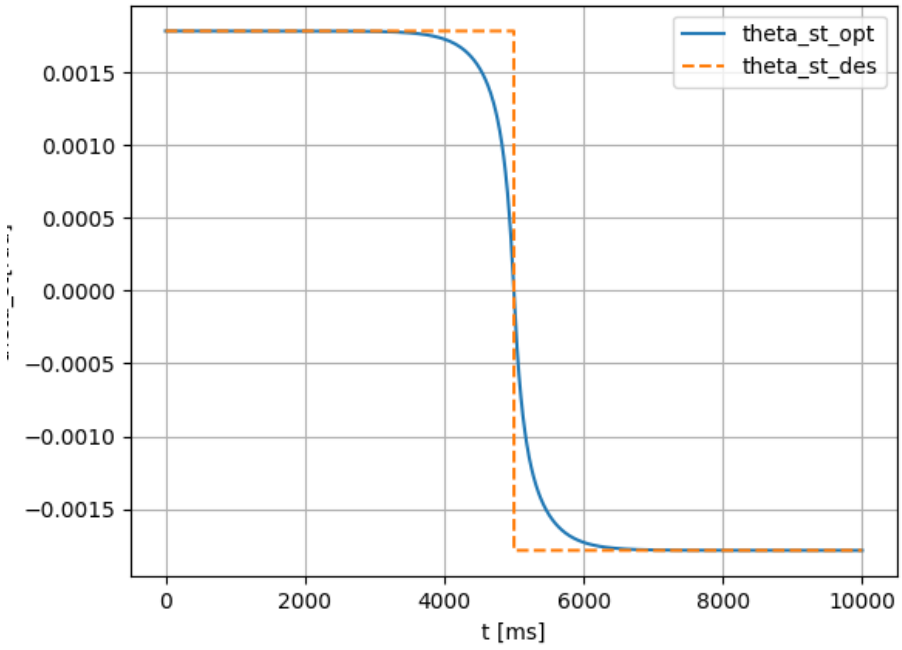


Figure 2.4: Step transition between θ_{st_1} and θ_{st_2}

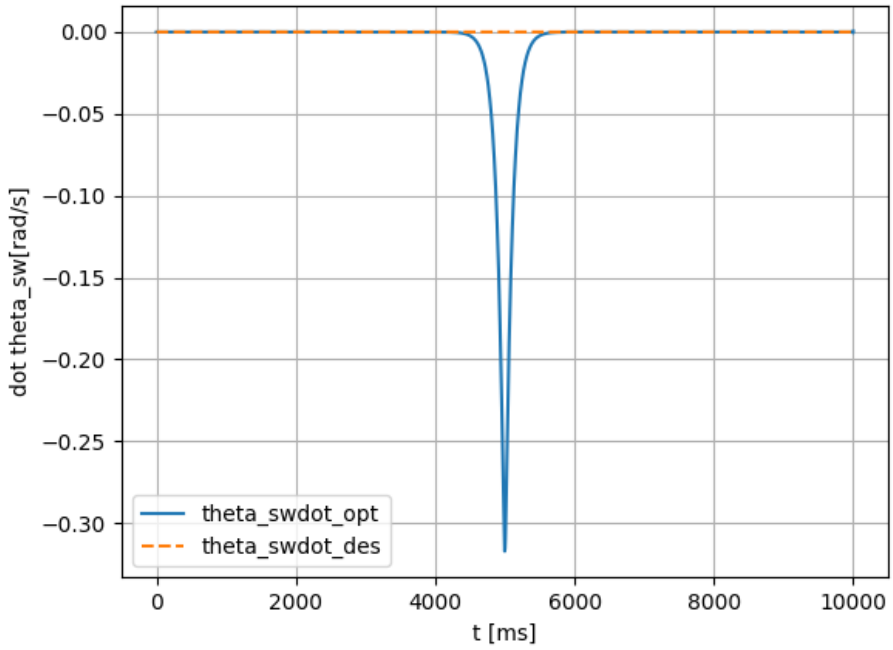


Figure 2.5: Angular velocity transition $\dot{\theta}_{sw}$

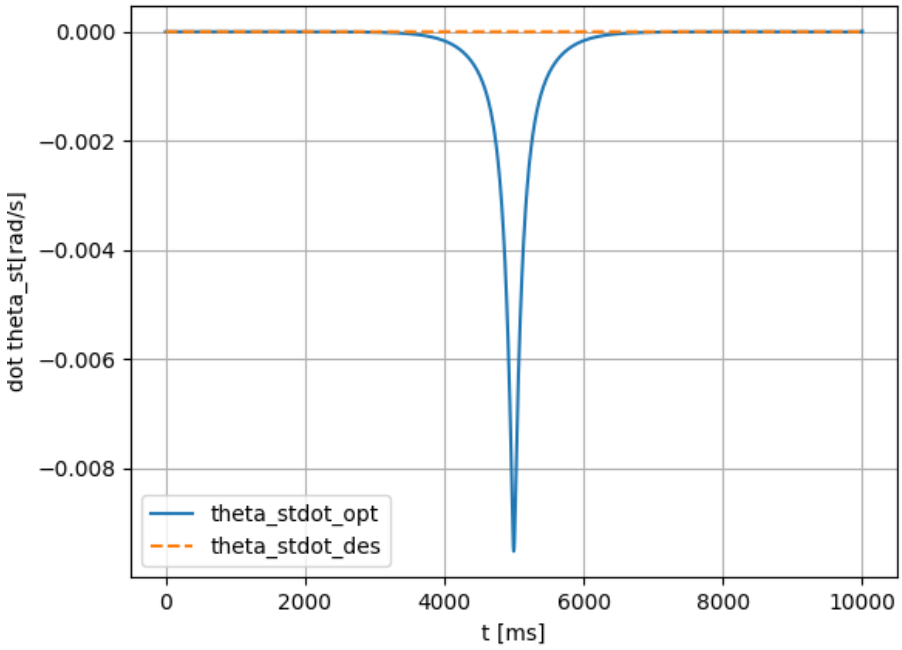


Figure 2.6: Angular velocity transition $\dot{\theta}_{st}$

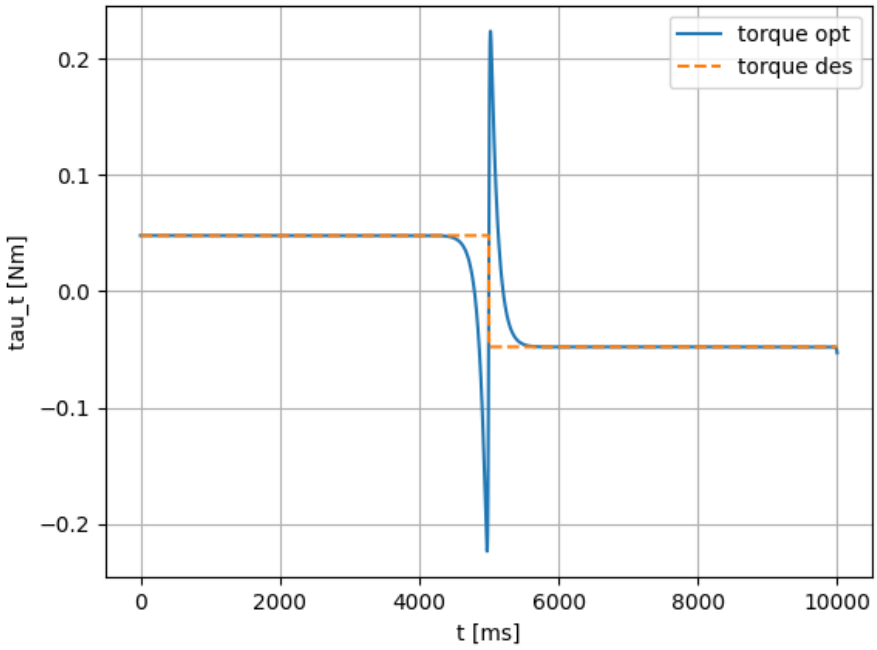


Figure 2.7: Torque evolution

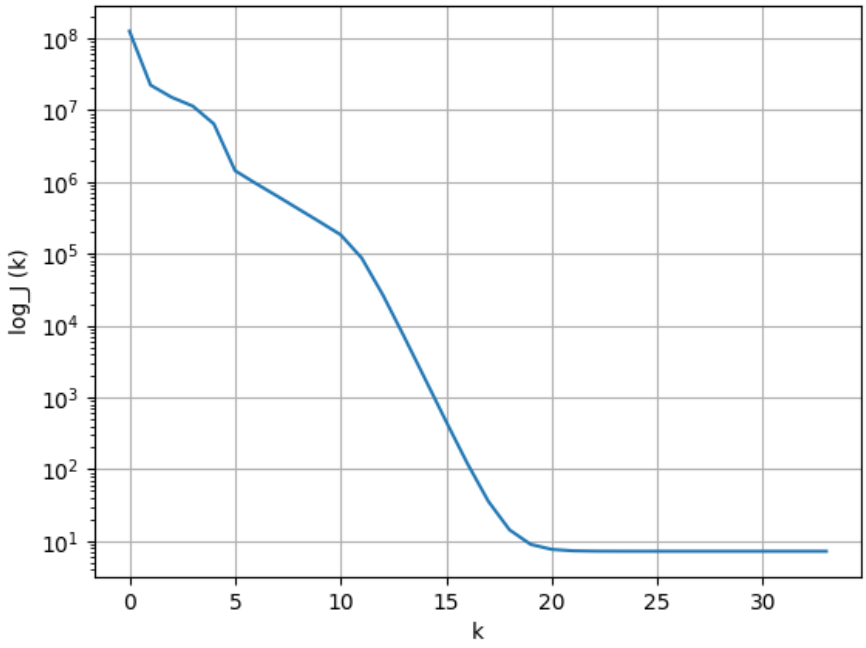


Figure 2.8: Cost evolution

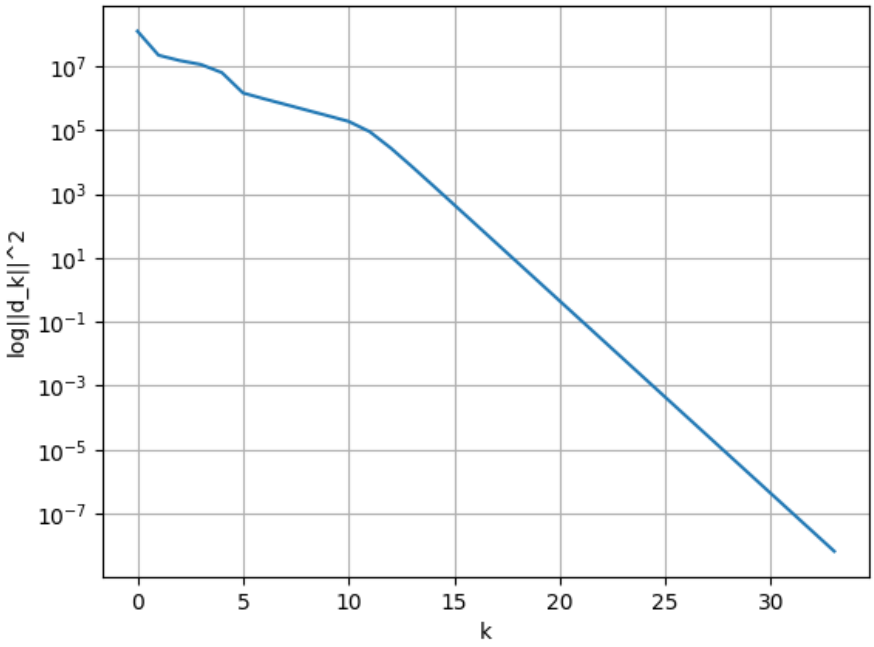


Figure 2.9: Descend evolution

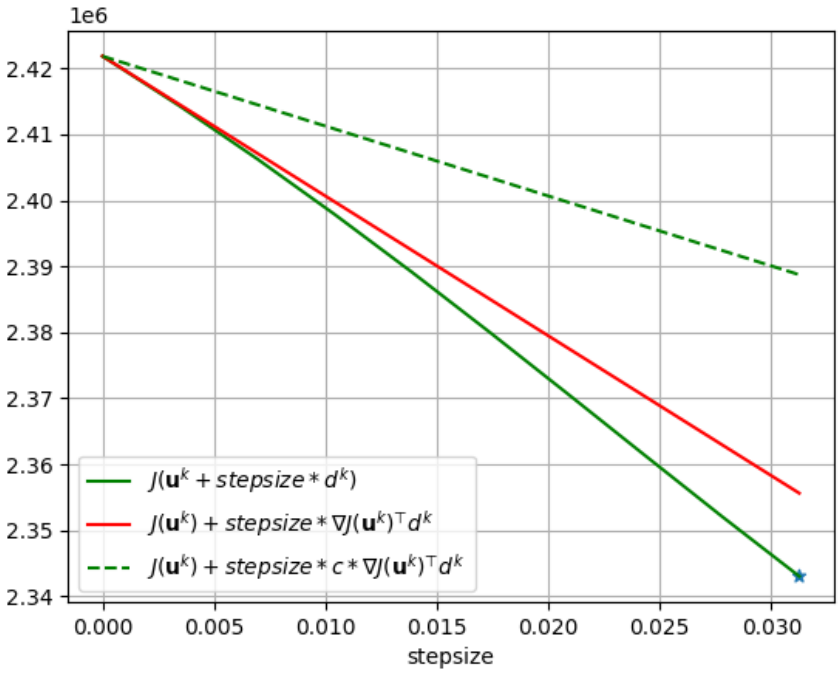


Figure 2.10: Descent direction after 5 itarations

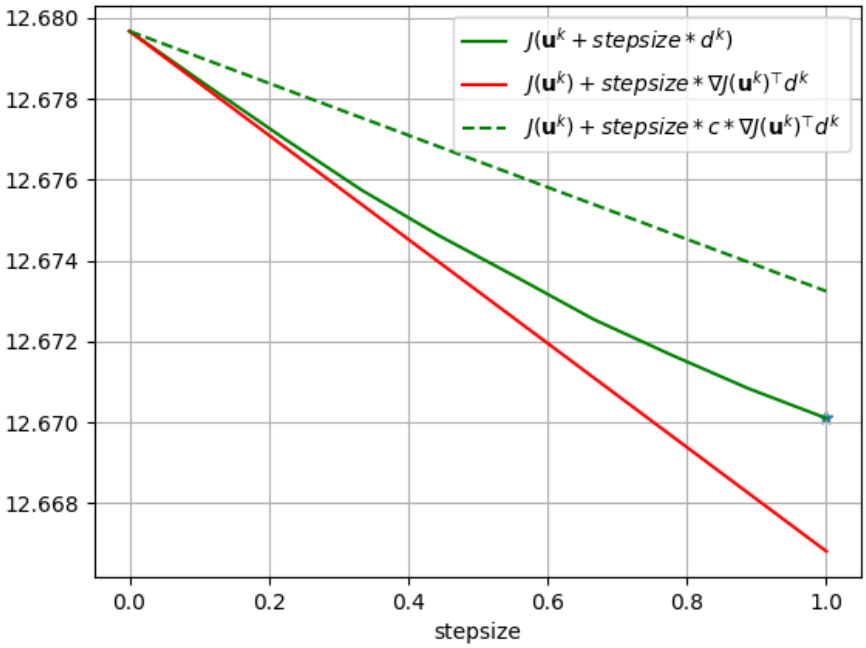


Figure 2.11: Descend direction after 12 iterations

2.2.2 Transition as smooth reference between equilibria

In this case of smooth reference, we choose to implement the sigmoid function where this change the behaviour of the transition a step, to the smooth reference trajectories, which guide the system toward a desired state.

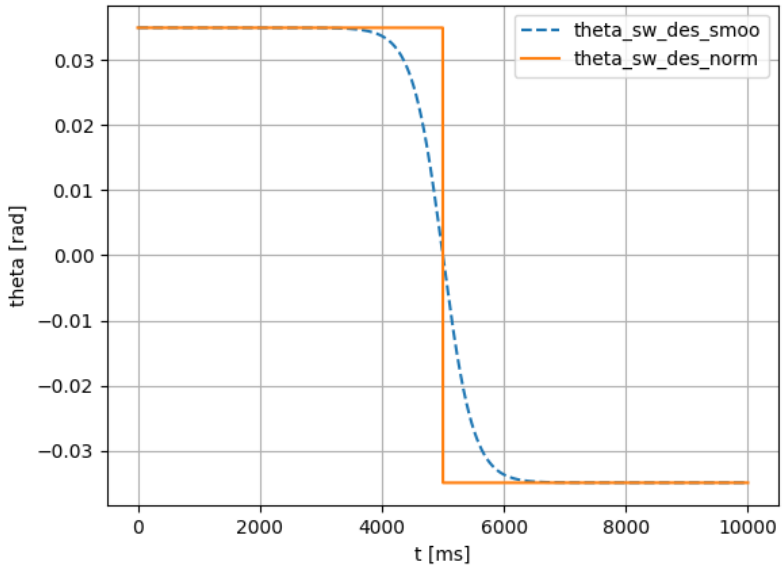


Figure 2.12: smooth trajectory connect the two θ_{sw} of equilibrium points

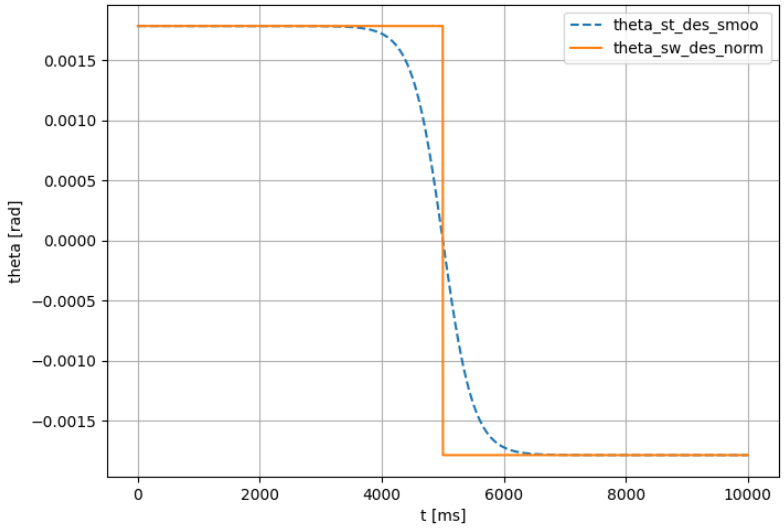


Figure 2.13: smooth trajectory connect the two θ_{st} of equilibrium points

Chapter 3

Trajectory optimization: smooth trajectory - Task 2

To represent the desired motion of a system over time it is referred to as a smooth reference trajectory. A reference trajectory is utilized as a target for the system to follow in optimal control and may be computed in a number of ways. In order to find the optimal trajectory that minimizes a cost function, one such technique is the Newton's algorithm for optimum control, which includes solving a nonlinear optimization problem.
we can write our optimization problem as:

$$\begin{aligned} \min_{\bar{x},\bar{u}} \quad & \bar{l}(\bar{x},\bar{u}) \\ \text{subj to } & x_{t+1} = f\left(x_t, u_t\right) \\ & x_0 = x_{\text{init}} \end{aligned}$$

given:

$$\begin{aligned} \bar{l}(\bar{x},\bar{u}) &= l_t(\bar{x},\bar{u}) + l_T\left(x_T\right) \\ &= \sum_{t=0}^{T-1} l_t\left(x_t, u_t\right) + l_T\left(x_T\right) \end{aligned}$$

where we denote l_t is the the **stage cost** and l_T is the **terminal cost**.

Now, we can rewrite our optimization problem as :

$$\begin{aligned} \min_{x_t,u_t} \quad & \sum_{t=0}^{T-1} l_t\left(x_t, u_t\right) + l_T\left(x_T\right) \\ \text{subj to } & x_{t+1} = f\left(x_t, u_t\right) \\ & x_0 = x_{\text{init}} \end{aligned}$$

In order to define our Cost Function we will use: a **Quadratic Cost Function** that is defined by:

$$l_t\left(x_t, u_t\right) = \frac{1}{2}\left(x_t - x_t^{des}\right)^\top Q_t \left(x_t - x_t^{des}\right) + \frac{1}{2}\left(u_t - u_t^{des}\right)^\top R_t \left(u_t - u_t^{des}\right) \tag{3.1}$$

$$l_T(x_T) = \frac{1}{2}\left(x_T - x_T^{des}\right)^\top Q_T \left(x_T - x_T^{des}\right) \tag{3.2}$$

Applying the first and second derivatives for our cost functor we will get:

- first derivatives:

$$\begin{aligned} \nabla_{x_t} l_t\left(x_t, u_t\right) &= Q_t\left(x_t - x_t^{des}\right) = q_t \\ \nabla_{u_t} l_t\left(x_t, u_t\right) &= R_t\left(u_t - u_t^{des}\right) = r_t \\ \nabla_{x_T} l_T\left(x_T\right) &= Q_T\left(x_T - x_T^{des}\right) = q_T \end{aligned} \tag{3.3}$$

- second derivatives:

$$\begin{aligned} \nabla^2_{x_t x_t} l_t\left(x_t, u_t\right) &= Q_t \\ \nabla^2_{u_t u_t} l_t\left(x_t, u_t\right) &= R_t \\ \nabla^2_{x_T x_T} l_T\left(x_T\right) &= Q_T \end{aligned} \tag{3.4}$$

- Mixed derivatives:

$$\begin{aligned} \nabla^2_{x_t u_t}\left(x_t, u_t\right) &= 0 = S_t \\ \nabla^2_{u_t x_t}\left(x_t, u_t\right) &= 0 = S_t^\top \end{aligned} \tag{3.5}$$

this lead to the following Optimization problem :

$$\min_{\Delta x_t, \Delta u_t} \sum_{t=0}^{T-1} \begin{bmatrix} q_t \\ r_t \end{bmatrix}^\top \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + \frac{1}{2}\begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}^T \begin{bmatrix} Q_t & S_t^\top \\ s_t & R_t \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + q_T^T x_T + \frac{1}{2}\Delta x_T^T Q_T \Delta x_T$$

$$\begin{aligned} \text{subj. to} \quad & \Delta x_{t+1} = f\left(\Delta x_t, \Delta u_t\right) \\ & \mathrm{x}_0 = x_{\text{init}} \end{aligned} \tag{3.6}$$

3.1 Newton's method optimal algorithm

The Newton's method uses first and second order derivatives to solve optimization problems in a specific recursive manner. However, in this case, we put it into practice in a way that is appropriate for optimal control and lets us take use of the fact that the constraint equation is just the system dynamic.

The method is then performed again until it reaches the correct minimum; for this reason, we'll think of the generic iteration k.

3.2 Initialization

If a suitable beginning trajectory is provided, the algorithm operates faster and more effectively. It must respect the dynamics of the system and begin at the same fixed point while reflecting the desired behavior. We made the decision to permanently address the initialization problem at the design stage in order to create a generic approach that is as case independent as possible:

- $u_t^0 = u_t^{\text{des}} \; \forall t = 0, 1, \ldots, T-1$ the initial input guess trajectory shall coincide with the reference curve
- $x_0^k = x_0^{\text{des}} \; \forall k = 0, 1, \ldots, N_{\max}$ the starting state point for all k state trajectory shall coincide with the starting point of the reference curve
- $x_{t+1}^0 = f\left(x_t^0, u_t^0\right) \quad \forall t = 1, 2, \ldots, T$ the rest of the initial guess state trajectory will be one obtained integrating the system dynamics in open loop from the starting point and using the desired input sequence

In this way we merged the initialisation problem with the one of finding a suitable desired reference curve.

3.3 Descend direction of iteration k

At the beginning of iteration k, we must evaluate the gradients of the dynamic and cost functions and we must evaluate them for each instant t of our time span T.

Although it is essential for the algorithm to operate properly, we are unable to choose the dynamic function's second order derivatives, and as a result, we cannot force them to be positive definite. To compute the descent direction there are two different ways one that mimics first order techniques and another that fully utilizes dynamic hessian could be one possibility.

We will initially utilize the first framework as a wide microscope focus to get near to the answer, and then we will switch to the second structure as a fine microscope focus. [Znegui et al.(2020b)Znegui, Gritli, and Belghith]

3.3.1 First order method

In the first scenario, the adjoint equation is not necessary, and we may immediately compute the row components of the subsequent optimization problem:

- $\mathbf{q}_t^k = \nabla_{x_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$ for $t \in [0, 1, \ldots T-1]$

- $\mathbf{r}_t^k = \nabla_{u_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$ for $t \in [0, 1, \dots T-1]$
- $\mathbf{q}_T^k = \nabla \ell_T \left(\mathbf{x}_T^k\right)$ dimension n
- $Q_t^k = \nabla^2_{x_t x_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$ for $t \in [0, 1, \dots T-1]$
- $S_t^k = \nabla^2_{x_t u_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$ for $t \in [0, 1, \dots T-1]$
- $R_t^k = \nabla^2_{u_t u_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$ for $t \in [0, 1, \dots T-1]$
- $Q_T^k = \nabla^2_{x_T x_T} \ell_T \left(\mathbf{x}_T^k\right)$ dimension (nxn)
- $A_t^k = \nabla_{x_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)^T$ for $t \in [0, 1, \dots T-1]$
- $B_t^k = \nabla_{u_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)^T$ for $t \in [0, 1, \dots T-1]$

3.3.2 Second order method

In this alternate scenario, we must first work backwards from the end of the adjoint equation to solve it:

$$\lambda_T^k = \nabla \ell_T \left(\mathbf{x}_T^k\right)$$
$$\lambda_t^k = \nabla_{x_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) \lambda_{t+1}^k + \nabla_{x_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$$

Then we are ready to compute the row elements of the future optimization problem:

- $\mathrm{q}_t^k = \nabla_{x_t} \ell_t \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)$
- $\mathrm{r}_t^k = \nabla_{u_t} \ell_T \left(\mathrm{x}_t^k, \mathbf{u}_t^k\right)$
- $\mathbf{q}_T^k = \nabla \ell_t \left(\mathbf{x}_T^k\right)$
- $Q_t^k = \nabla^2_{x_t x_t} \ell_T \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) + \nabla^2_{x_t x_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) \cdot \lambda_{t+1}^k$
- $S_t^k = \nabla^2_{x_t u_t} \ell_T \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) + \nabla^2_{x_t u_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) \cdot \lambda_{t+1}^k$
- $R_t^k = \nabla^2_{u_t u_t} \ell_T \left(\mathrm{x}_t^k, \mathbf{u}_t^k\right) + \nabla^2_{u_t u_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right) \cdot \lambda_{t+1}^k$
- $Q_T^k = \nabla^2_{x_T} \ell_T \left(\mathbf{x}_T^k\right)$
- $A_t^k = \nabla_{x_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)^T$
- $B_t^k = \nabla_{u_t} f \left(\mathbf{x}_t^k, \mathbf{u}_t^k\right)^T$

3.3.3 Optimization problem statement

The descend direction can be formalised as an input step $\Delta \mathbf{u}_t$ we compute its whole time sequence as vector solution of the affine LQR optimization problem whose parameters were just obtained:

$$\min_{\Delta x_t^k, \Delta u_t^k} \sum_{t=0}^{T-1} \begin{bmatrix} q_t^k \\ r_t^k \end{bmatrix}^T \begin{bmatrix} \Delta x_t^k \\ \Delta u_t^k \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta x_t^k \\ \Delta u_t^k \end{bmatrix}^T \begin{bmatrix} Q_t^k & S_t^{kT} \\ S_t^k & r_t^k \end{bmatrix} \begin{bmatrix} \Delta x_t^k \\ \Delta u_t^k \end{bmatrix}$$
$$+\nabla l_T \left(x_T^k\right)^T \Delta x_T + \frac{1}{2} \Delta x_T^T Q_T^k \Delta x_T$$
$$\text{subj. to } \Delta x_{t+1}^k = A_t^k \Delta x_t^k + B_t^k \Delta u_t^k$$

- **Backward computation of the gain time-sequence**

  To solve the optimization problem we must run the following backward procedure to obtain the time-sequences related to iteration k of matrix K_t^* of dimension mxn, vector p_t of dimension m, matrix P_t of dimension nxn, vector p_t of dimension n. [Gritli et al.(2013)Gritli, Khraief, and Belghith]

  Start from $p_T^k = q_T$ and $P_T^k = Q_T$

  Compute for $t \in [T-1, T-2, \dots 1, 0]$:

$$K_t^* = -\left(R_t + B_t^T P_{t+1} B_t\right)^{-1} \left(S_t + B_t^T P_{t+1} A_t\right)$$
$$\sigma_t^* = -\left(R_t + B_t^T P_{t+1} B_t\right)^{-1} \left(r_t + B_t^T p_{t+1} + B_t^T P_{t+1} c_t\right)$$
$$p_t = q_t + A_t^T p_{t+1} + A_t^T P_{t+1} c_t + K_t^* \left(R_t + B_t^T P_{t+1} B_t\right) \sigma_t^*$$
$$P_t = Q_t + A_t^T P_{t+1} A_t - K_t^{*T} \left(R_t + B_t^T P_{t+1} B_t\right) K_t^*$$

- **Forward computation of the input steps**

  Now we are ready to compute forward in time the sequence of input descend directions and in order to do that we must update also a delta state sequence. Start from $\Delta x_0^k = 0$ and compute for $t \in [0, 1, \dots T-1]$

$$\Delta u_t^k = K_t^* \Delta x_t^k + \sigma_t^*$$
$$\Delta x_{t+1}^k = A_t \Delta x_t^k + B_t \Delta u_t^k$$

3.3.4 Trajectory update at iteration k

In light of the descend directions just obtained, we can compute the state-input trajectory of next iteration k + 1.

- **New input sequence**

  Here we should implement a step size selection and the Armijo rule has proved itself to be extremely useful to make sure that update would bring a real drop in the cost function. With the suitable γ^k computed implementing the Armijo rule we can compute for $t \in [0, 1, \dots T-1]$:

$$u_t^{k+1} = u_t^k + \gamma \Delta u_t^k$$

- **New state trajectory**

  we forward integrate the system dynamics, start from $x_0^k = x_{\text{init}}$ then compute for all $t \in [0, 1, \ldots T-1]$:

$$x_{t+1}^{k+1} = f\left(x_t^{k+1}, u_t^{k+1}\right)$$

The optimal control input that steers the system from its current state to the desired state in the shortest amount of time is found using the Newton's algorithm for optimal control. It is based on the principles of optimization and selects the best control input based on the gradient of the performance index. The sigmoid function may be utilized in this situation to build a smooth reference trajectory that aids the control system in smoothly and effectively following the target path. [Bertsekas(1997)]

The Newton's algorithm can be utilized to solve the optimization problem. The parameters are updated in the direction of steepest descent after computing the gradient of the cost function with respect to the parameters at each iteration. When the gradient is almost zero, the procedure is complete and the optimal parameters have been found.

After the reference trajectory has been established as a sigmoid function, the optimization issue may be expressed as finding the best values for the parameters that minimize a cost function. One of the characteristics that may be utilized to choose the cost function is stability and smoothness.

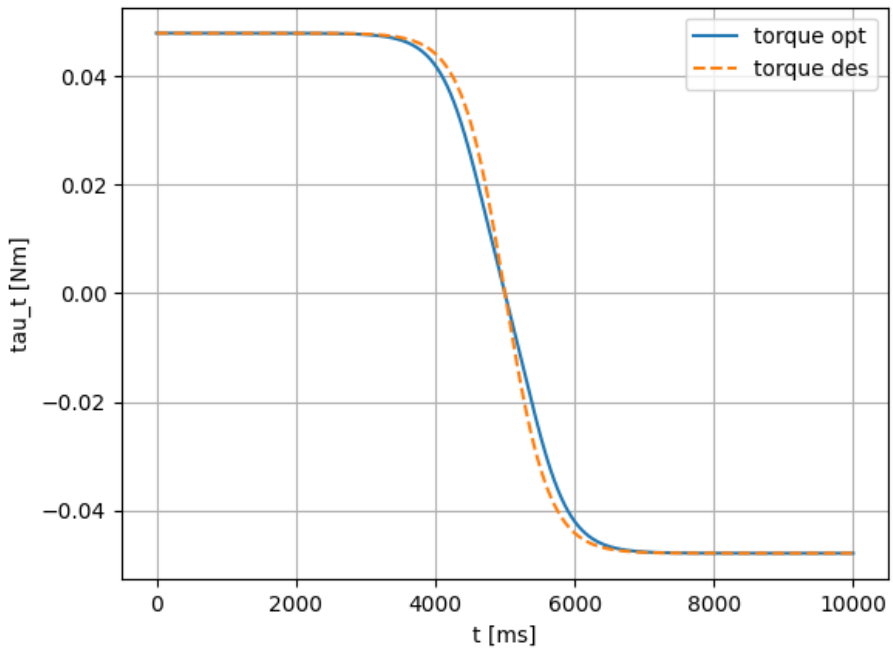


Figure 3.1: Optimal input torque need to perform our desired trajectory

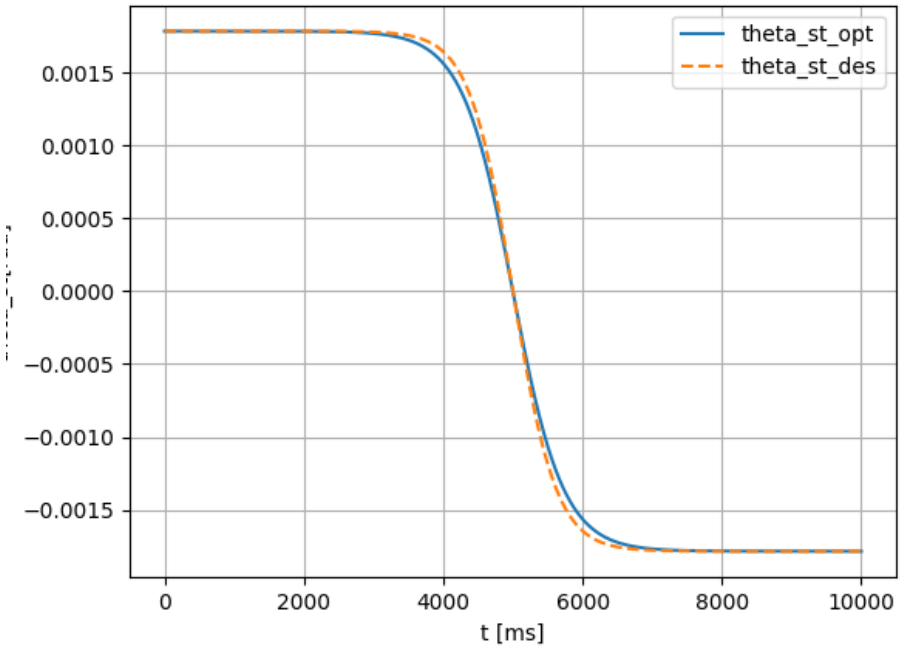


Figure 3.2: Optimal stance angle position after smoothing

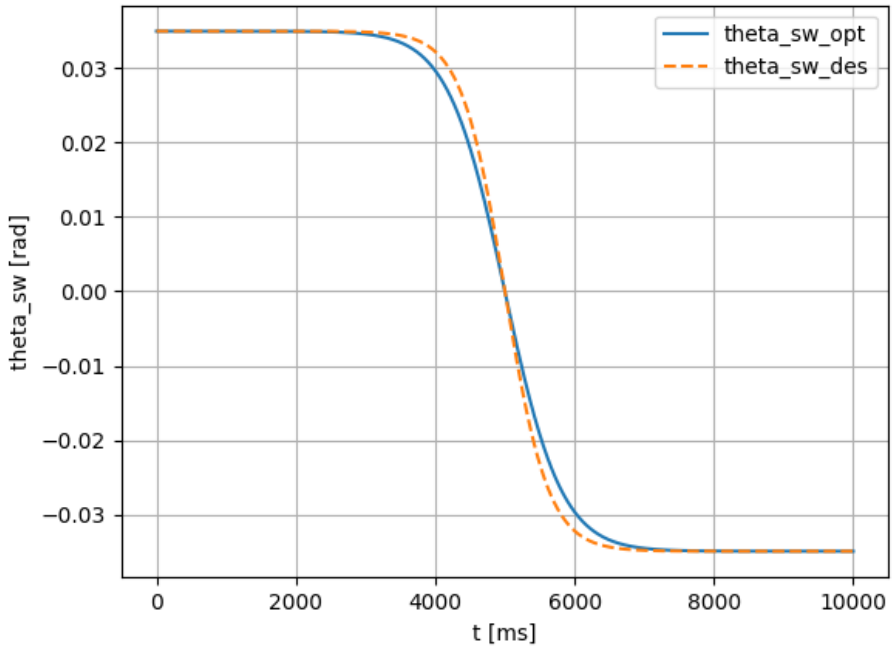


Figure 3.3: Optimal swing angle position after smoothing

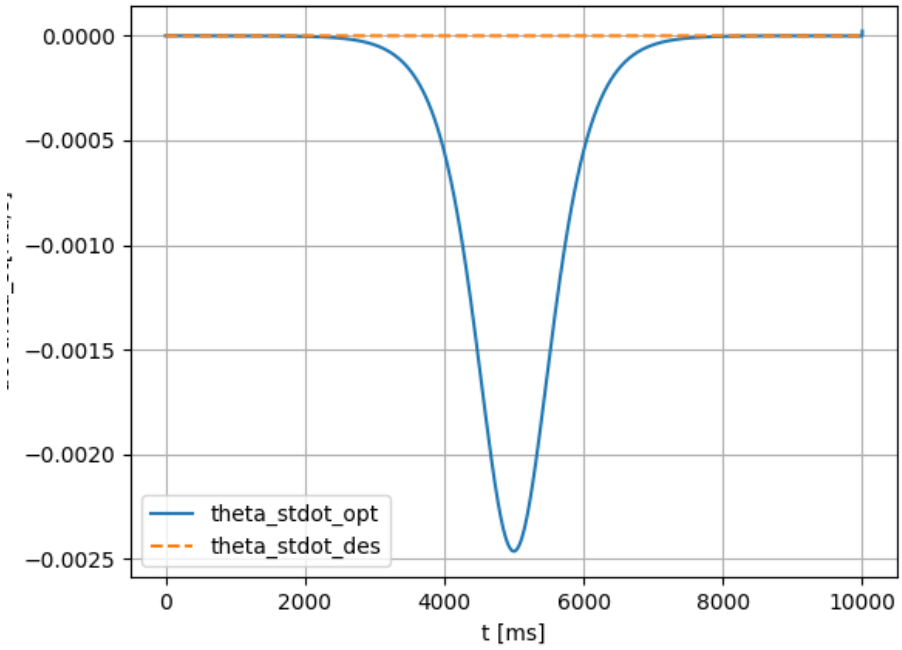


Figure 3.4: Optimal stance angular velocity to reach the second equilibrium point

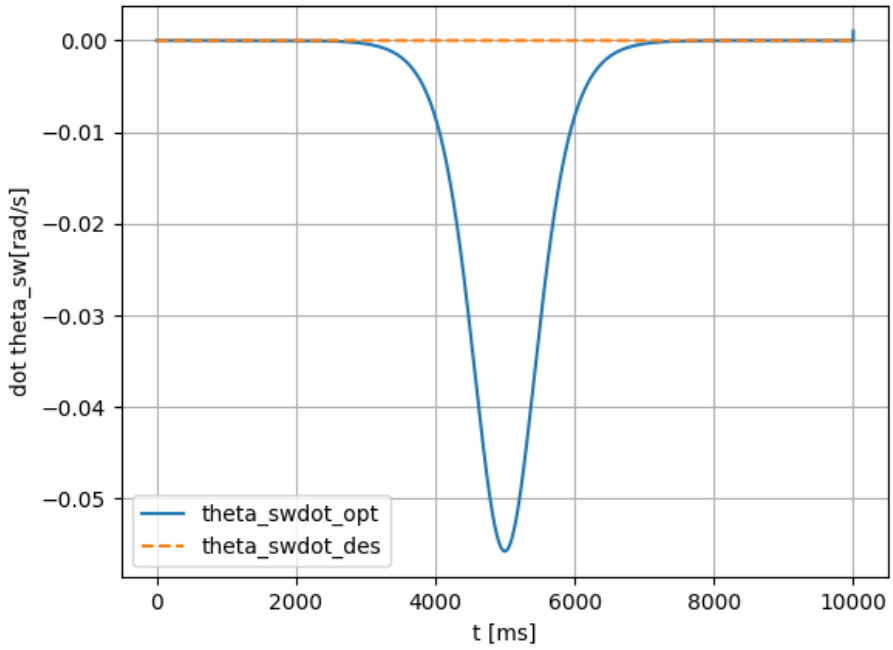


Figure 3.5: Optimal swing angular velocity to reach the second equilibrium point

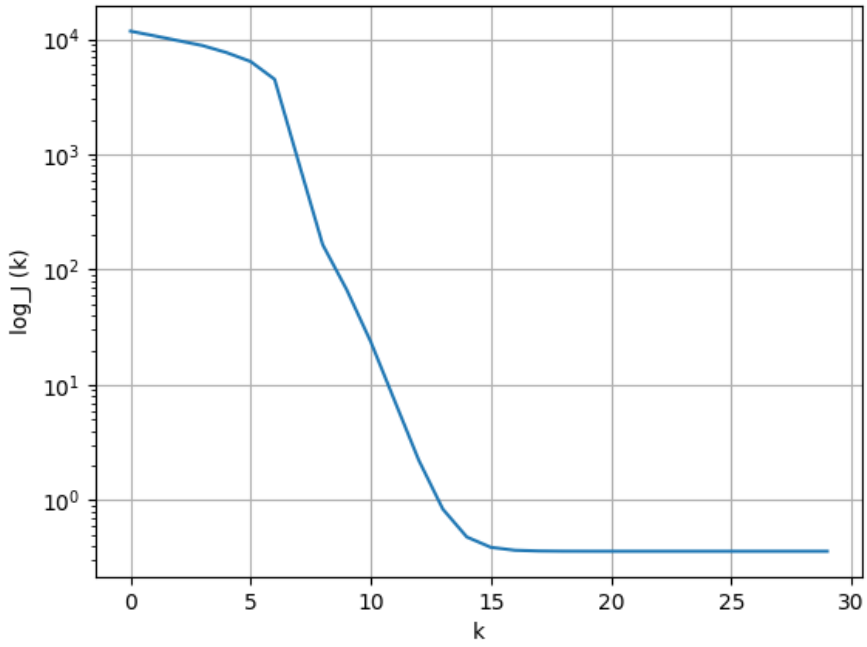


Figure 3.6: Cost evolution during the search phase for the optimal trajectory

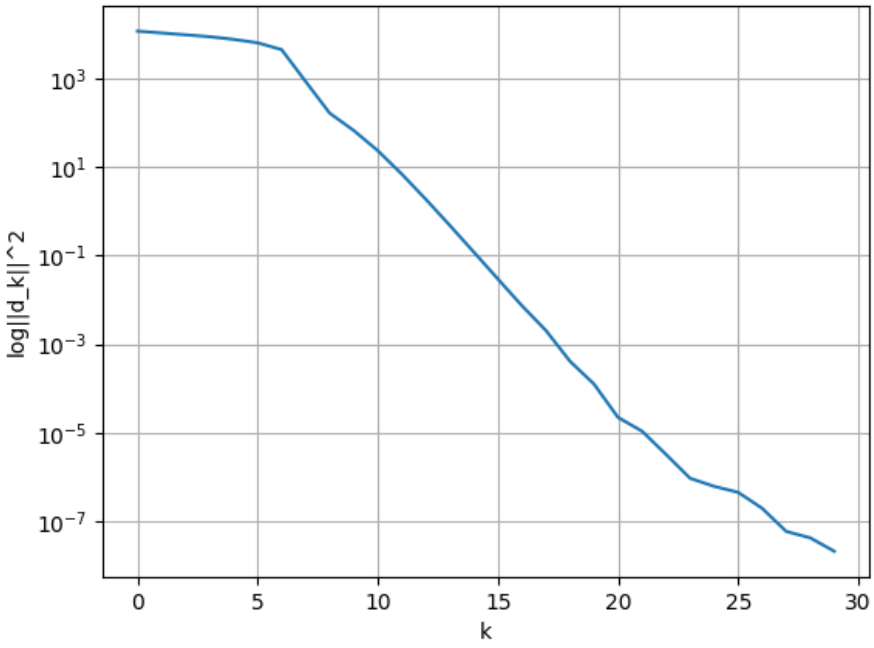


Figure 3.7: Descend norm evolution during the search phase for the optimal trajectory

Chapter 4

Trajectory tracking - Task 3

4.1 Trajectory tracking

The results of the previous task, in which we computed the optimal trajectory (x*; u*) for the system dynamics, are built upon in the third task. We linearize the system dynamics around this optimal trajectory to further improve the system's control performance. By linearizing the nonlinear system dynamics, we may convert it into a linear model that is more controllable and simpler to operate [Goswami et al.(1996)Goswami, Thuilot, and Espiau]. As a result, the third task is essential for maintaining the system's stability and desired performance. We are able to create a feedback controller that can precisely track the reference trajectory and successfully regulate the system by linearizing the system dynamics and using the LQR approach.

$$\Delta \mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_t^* \quad \text{for} \quad t \in [0, 1, \dots T]$$

$$\Delta \mathbf{u}_t = \mathbf{u}_t - \mathbf{u}_t^* \quad \text{for} \quad t \in [0, 1, \dots T-1]$$

$$\min_{\Delta x_t^k, \Delta u_t^k} \quad \sum_{t=0}^{T-1} \frac{1}{2}(\Delta \mathbf{x}_t^T Q_t \Delta \mathrm{x}_t + \Delta \mathbf{u}_t^T R_t \Delta \mathbf{u}_t) + \frac{1}{2}\Delta \mathbf{x}_T^T Q_T \Delta x_T$$
$$\text{subj. to } \Delta x_{t+1}^k = A_t^* \Delta x_t^k + B_t^* \Delta u_t^k \qquad \text{for } t \in [0, 1, \dots T-1]$$

The solution of the LQR problem gave us as result the optimal sequence of gain K. the we update the input as:

$$u_t = u_t^* + K_t^* \left(x_t - x_t^*\right) \tag{4.1}$$

with

$$\Delta(x_0) = x_{init}$$

then we find our trackng trajectory.
The result of that implementation can be seen in the following pages

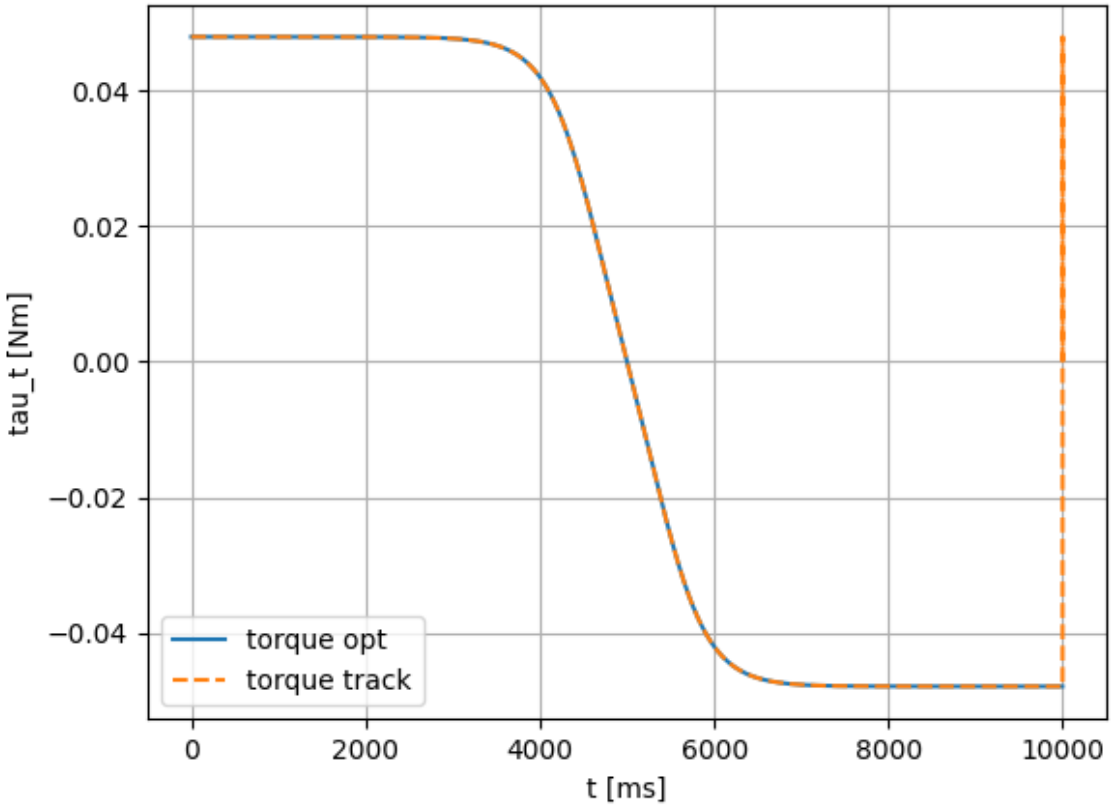


Figure 4.1: Optimal input vs tracking

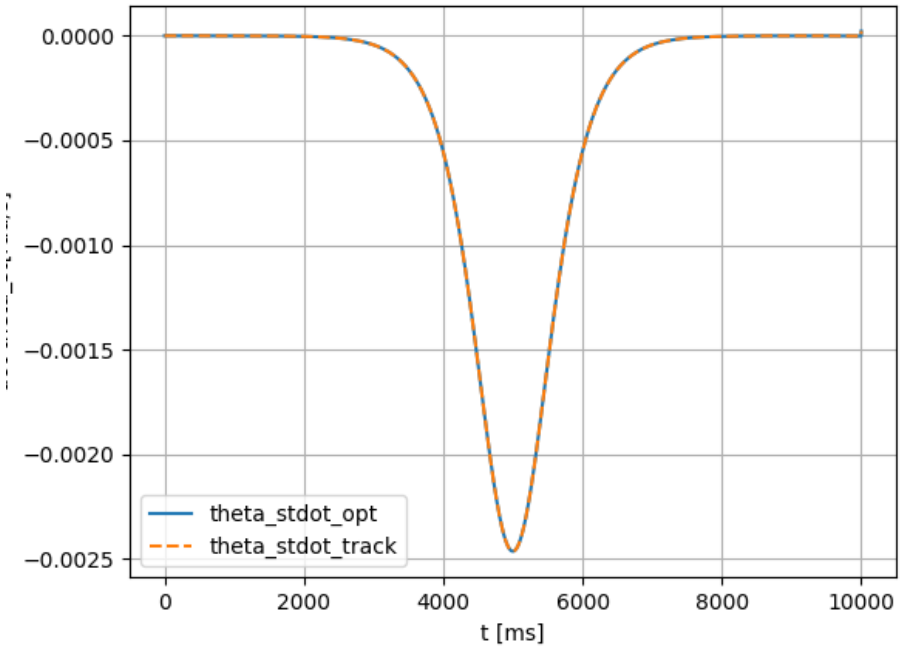


Figure 4.2: Optimal $\dot{\theta}_{st}$ vs tracking

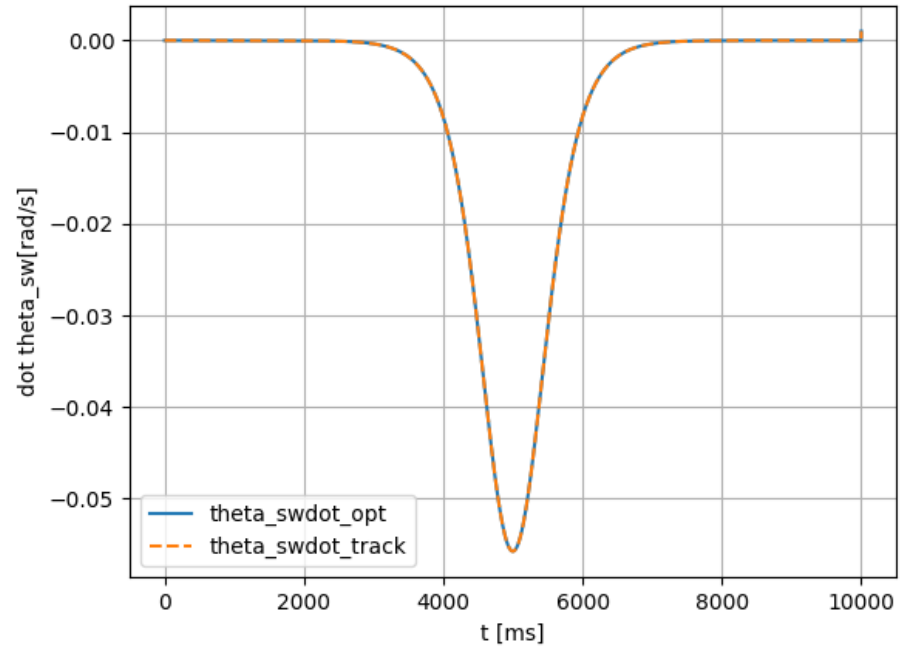


Figure 4.3: Optimal $\dot{\theta}_{sw}$ vs tracking

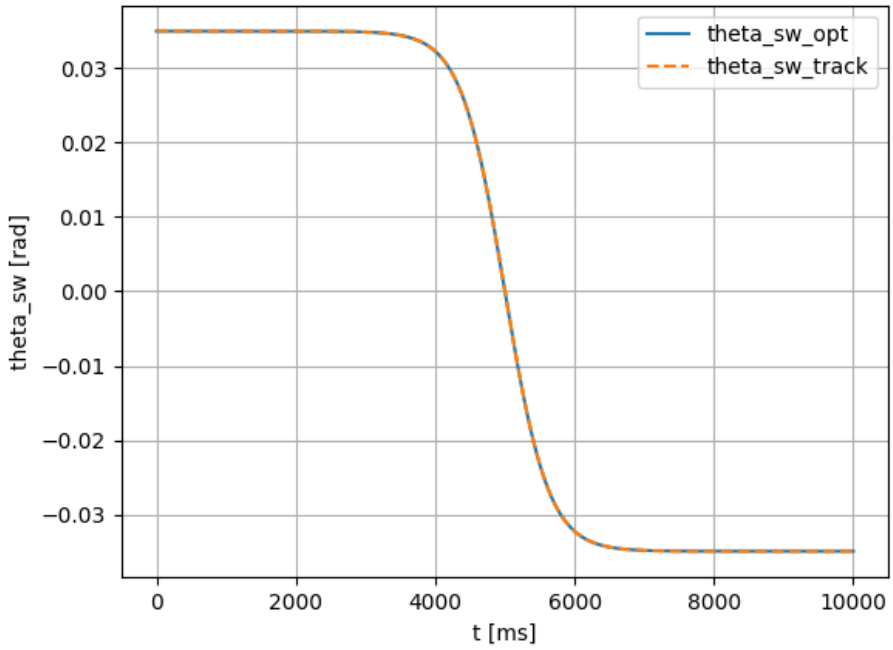


Figure 4.4: Optimal θ_{sw} vs tracking

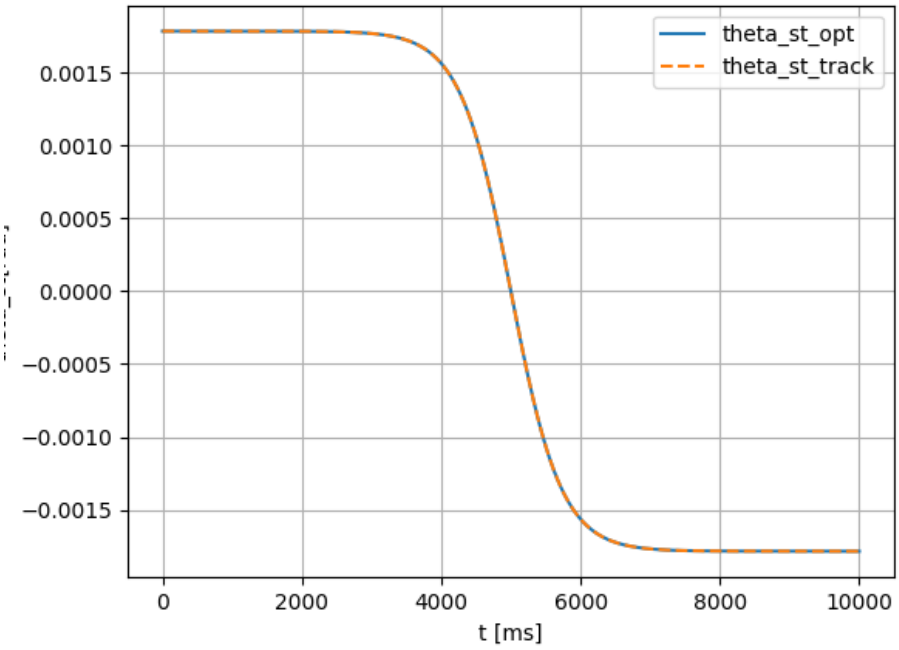


Figure 4.5: Optimal θ_{st} vs tracking

4.2 Noise simulation and robustness

4.2.1 Different initial position with respect to the desired trajectory

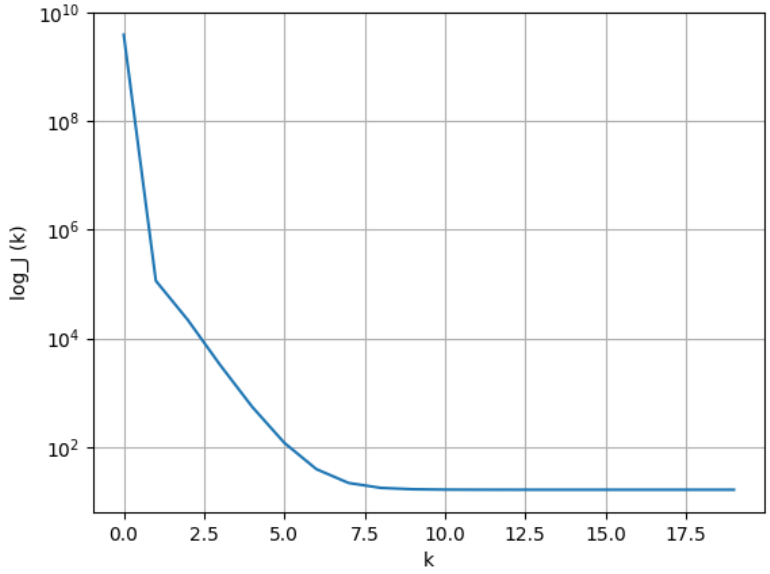


Figure 4.6: Cost function evolution

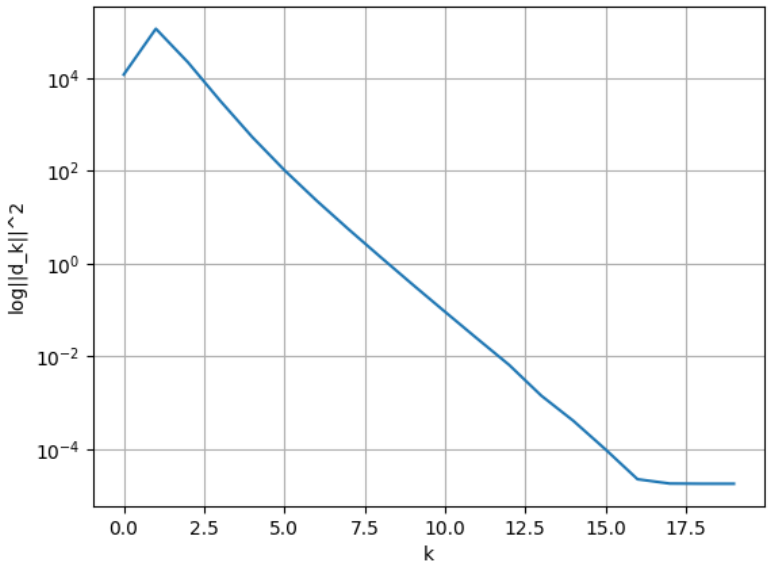


Figure 4.7: Descend norm evolution

4.2.2 Introduction of noise during trajectory tracking

After computing the desired best trajectory and writing an algorithm to follow it, we try to introduce in our system some disturbance in order to see how it responds and evaluate the efficiency of the programm control. for this work we perturbed the initial start condition of the leg to follow the optimal trajectory.

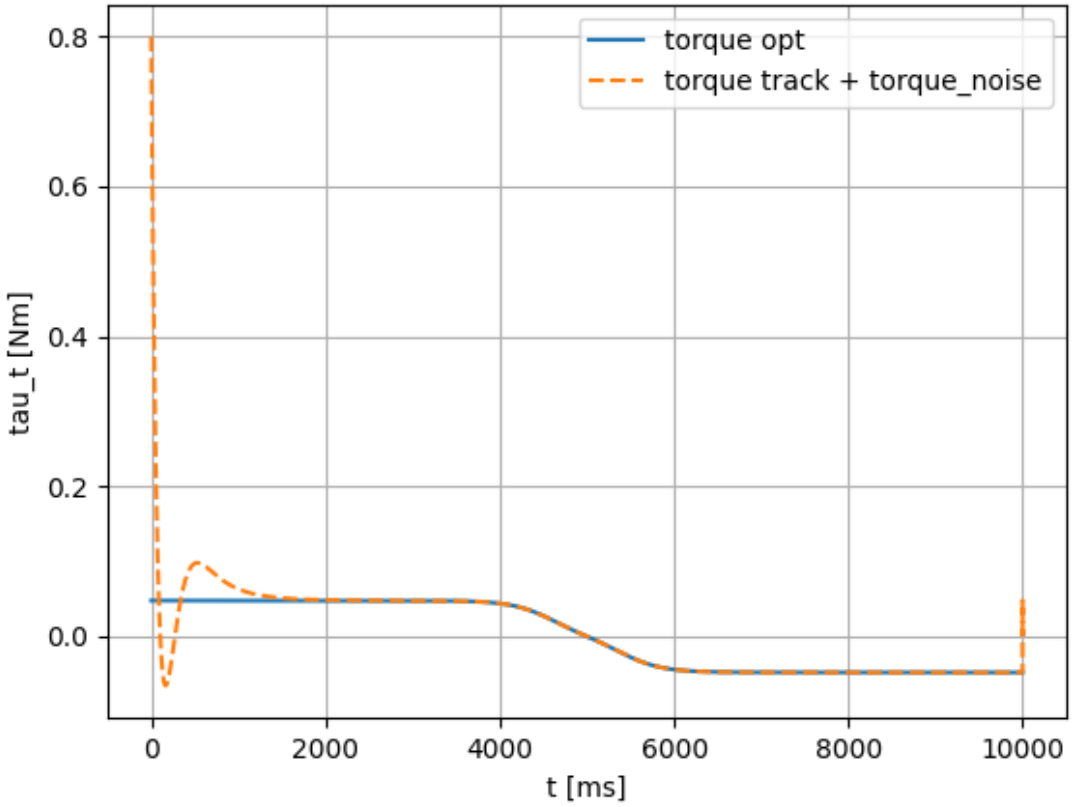


Figure 4.8: Optimal input vs noise effect

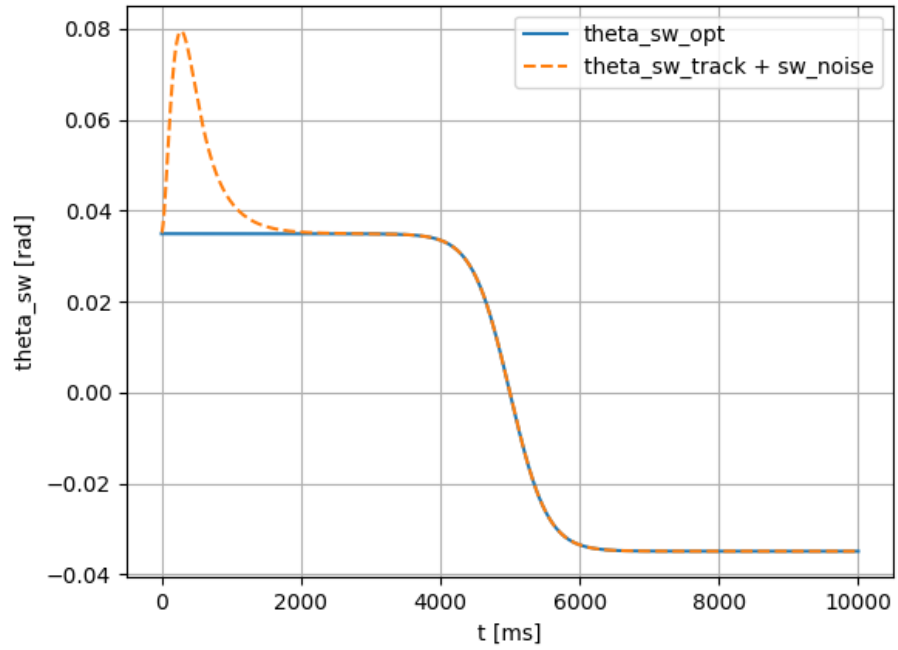


Figure 4.9: Optimal θ_{sw} vs noise effect

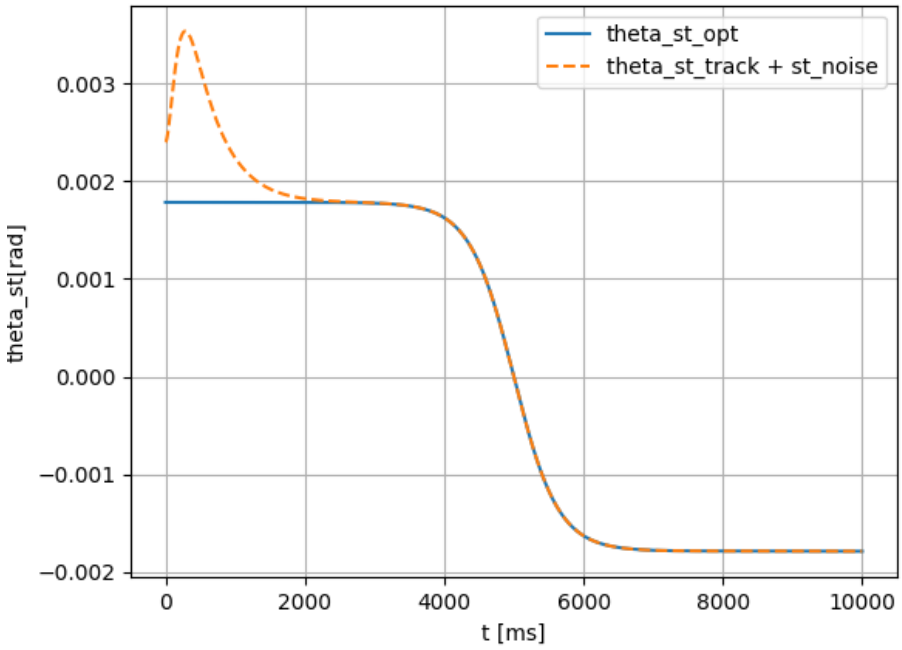


Figure 4.10: Optimal θ_{st} vs noise effect

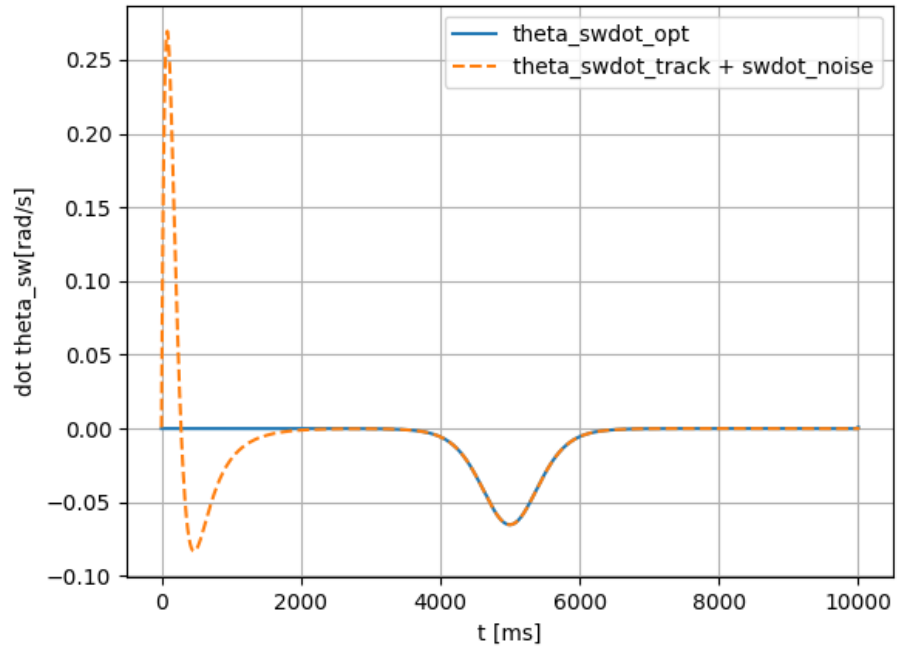


Figure 4.11: Optimal $\dot{\theta}_{sw}$ vs noise effect

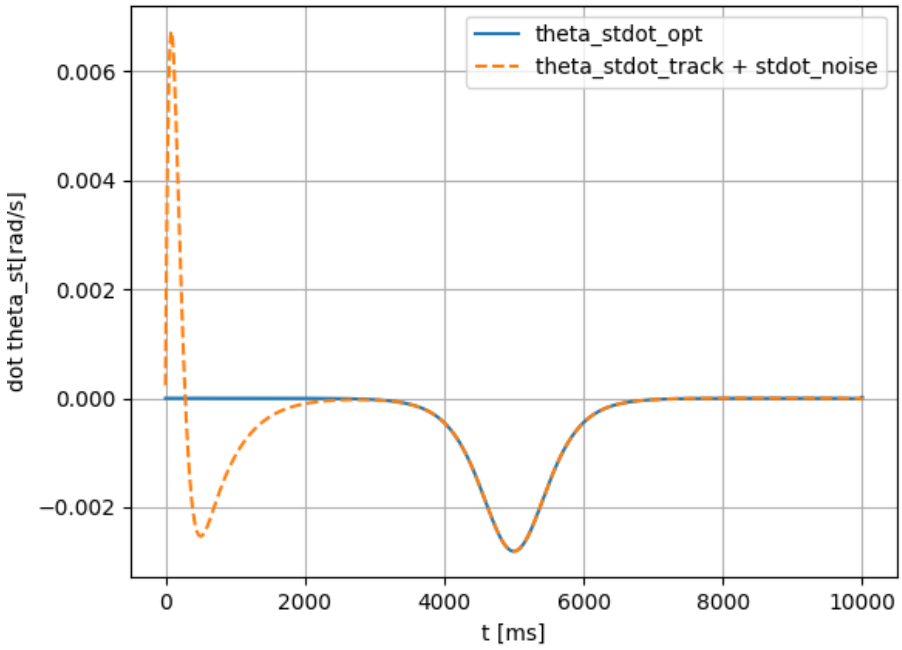


Figure 4.12: Optimal $\dot{\theta}_{st}$ vs noise effect

Chapter 5

Simulation part

One of the difficulties in making a legged robot walk is keeping it balance where should place its feed.This difficulty comes from the fact that contact forces with the environment are an absolutely necessity to generate and control leg.

In this part of project,given the simplified model of dynamics we are working with,we limited ourselves to simulating the movement of a leg from one position to another.If we try to ask to the robot to move forward,he sink in the ground due to the unmodel collision o contact force with the gorund at the of transistion.

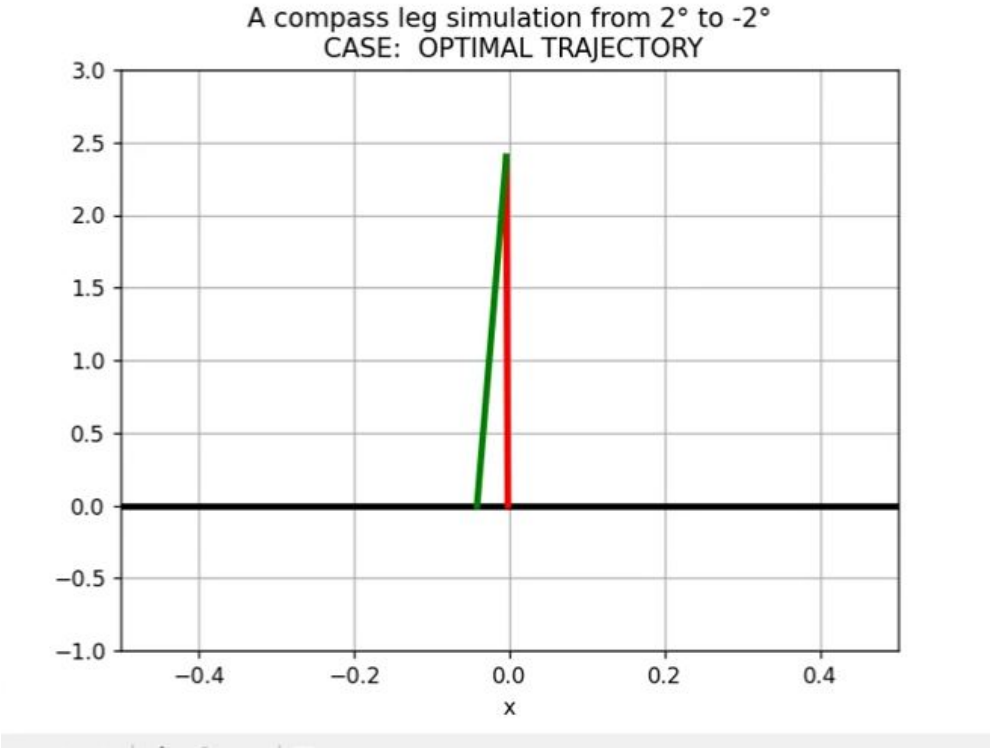


Figure 5.1: Compass leg simulation

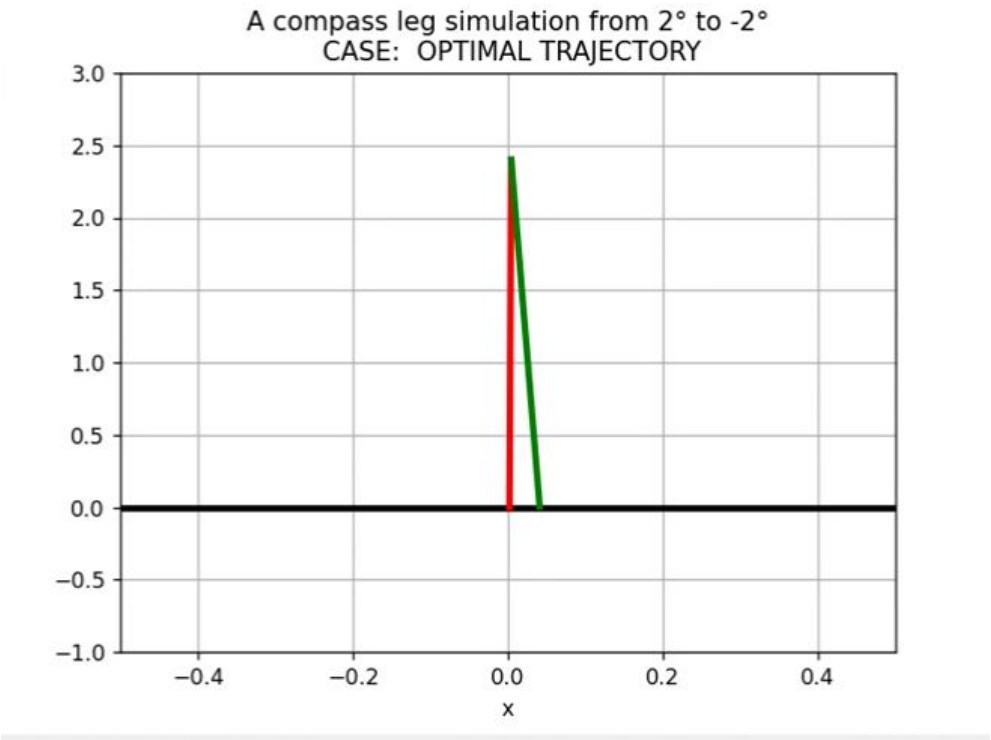


Figure 5.2: Compass leg simulation

Conclusions

Finally, numerical simulations demonstrate the effectiveness of the proposed framework. where, we present a theoretical study on gait planning and control of bipedal walking using the compass gait model. The project has been really energizing, and it allows us to work on a fascinating subject like the optimal control used with Humanoid Robots. The first obvious task related to this system has been to evaluate the stability of gait cycles as we have only explored efficiency of walking. We presented a robust method for generating a walking bipeds using numerical continuation methods (Newton's algorithm), and the design process for our model has being done iteratively using to make the control and minimize the cost. By altering the reference trajectory and using the one with the velocity profile, more advancements on this subject can be made

Bibliography

[Bertsekas(1997)] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[Ames and Poulakakis(2018)] A. D. Ames and I. Poulakakis, "Hybrid zero dynamics control of legged robots," 2018.

[Znegui et al.(2020a)Znegui, Gritli, and Belghith] W. Znegui, H. Gritli, and S. Belghith, "Stabilization of the passive walking dynamics of the compass-gait biped robot by developing the analytical expression of the controlled poincaré map," *Nonlinear Dynamics*, vol. 101, pp. 1061–1091, 2020.

[Znegui et al.(2020b)Znegui, Gritli, and Belghith] ——, "Design of an explicit expression of the poincaré map for the passive dynamic walking of the compass-gait biped model," *Chaos, Solitons & Fractals*, vol. 130, p. 109436, 2020.

[Gritli et al.(2013)Gritli, Khraief, and Belghith] H. Gritli, N. Khraief, and S. Belghith, "Chaos control in passive walking dynamics of a compass-gait model," *Communications in Nonlinear Science and Numerical Simulation*, vol. 18, no. 8, pp. 2048–2065, 2013.

[Goswami et al.(1996)Goswami, Thuilot, and Espiau] A. Goswami, B. Thuilot, and B. Espiau, "Compass-like biped robot part i: Stability and bifurcation of passive gaits," Ph.D. dissertation, INRIA, 1996.